Concurrent Programming

Algorithms

Merge Sort

Algorithm :

1. IF you are the root
   THEN load the array to be sorted

   ELSE get the array from the parent process

Merge Sort

2. IF the array has more than 2 elements AND (number of processes < max processes) [you can also add a condition like “array not already sorted”] THEN
   - create 2 processes and send each a (roughly equal) part of the array
   - wait for the two sorted arrays
   - merge them into one sorted array

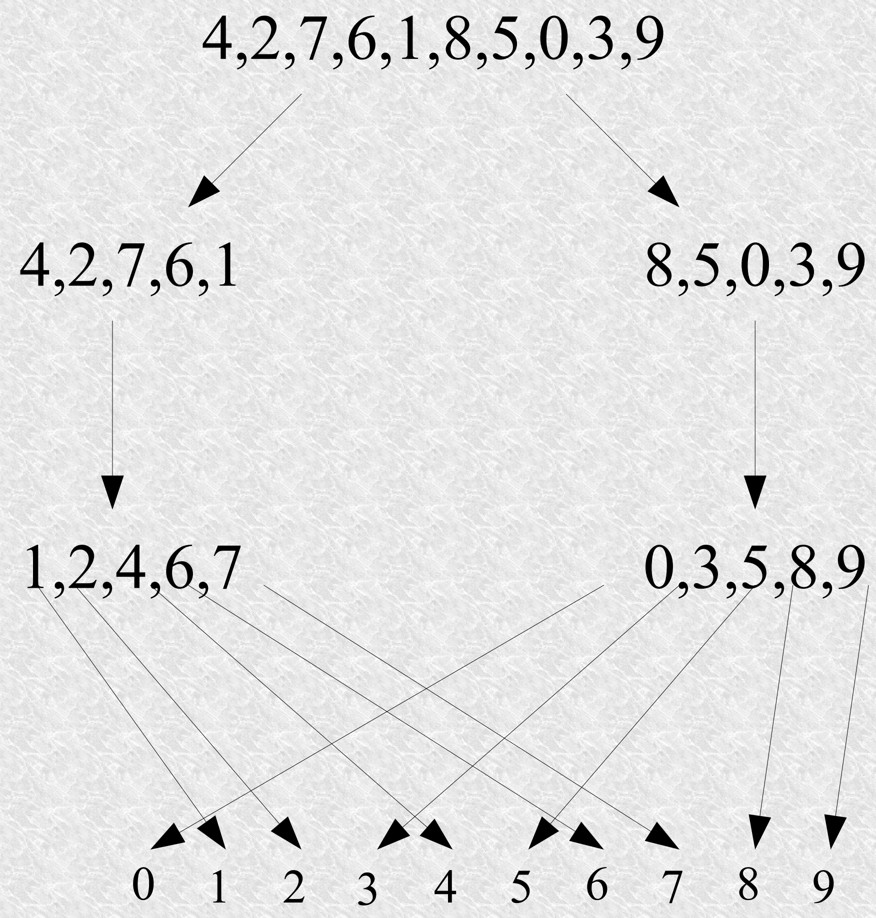
   ELSE sort the current array (in the simplest case, return 1 element or compare 2

Merge Sort

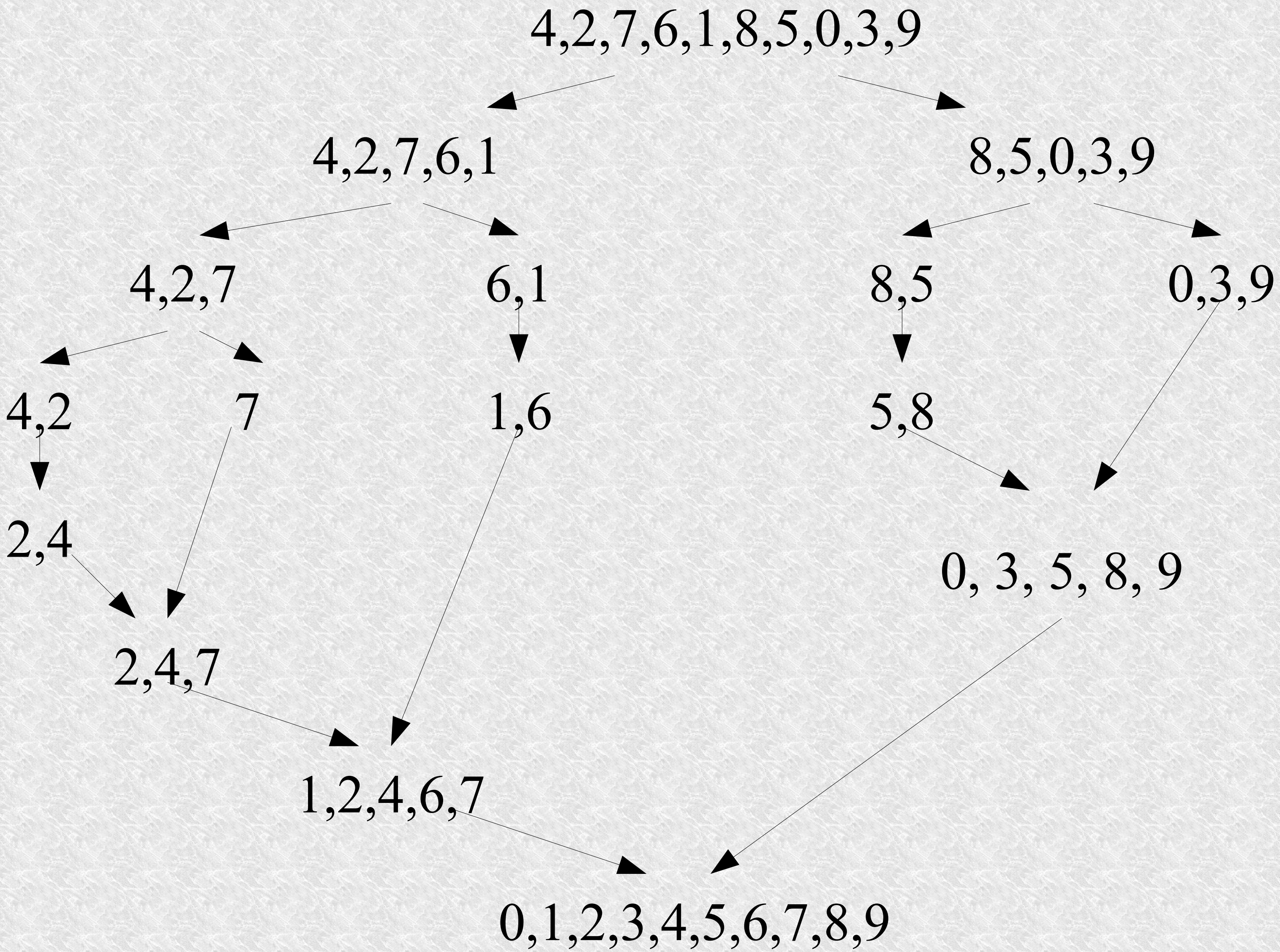
3. IF you are the root THEN display/save the result

   ELSE send the sorted array back to the parent

Merge Sort



Merge Sort

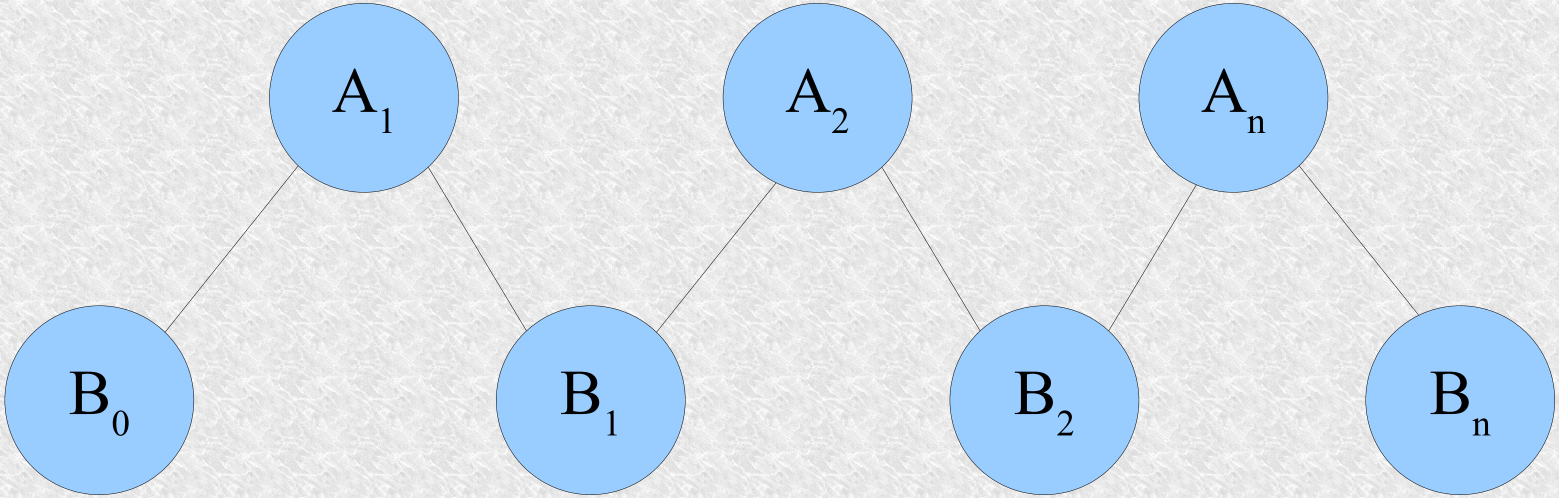

4,2,7,6,1,8,5,0,3,9
4,2,7,6,1
8,5,0,3,9
4,2,7
6,1
8,5
0,3,9
4,2
7
1,6
5,8
2,4
0, 3, 5, 8, 9
2,4,7
1,2,4,6,7
0,1,2,3,4,5,6,7,8,9

Oscillatory Sorting

For a sequence of length **n**, we create two sets of processes:

$\mathbf{A_1, A_2, \ldots A_n}$

$\mathbf{B_0, B_1, \ldots B_n}$



Oscillatory Sorting

A_i type tasks work as follows:

- Get 2 numbers
- the smaller one is sent to B_{i-1}
- bigger to B_i

Oscillatory Sorting

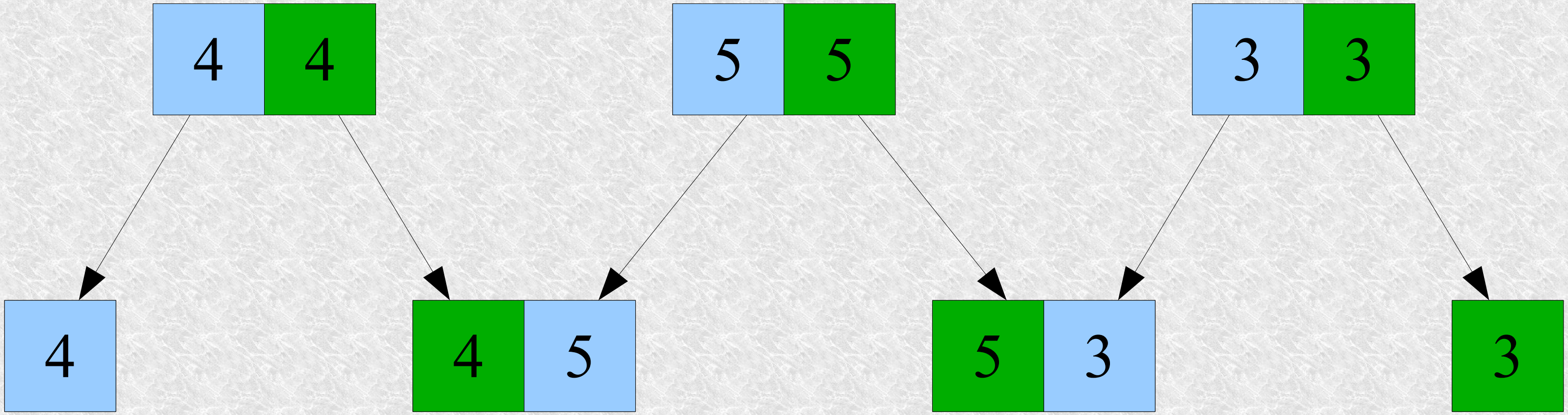
A_i type tasks work as follows:

- Get 2 numbers
- the smaller one is sent to A_i
- bigger to A_{i+1}

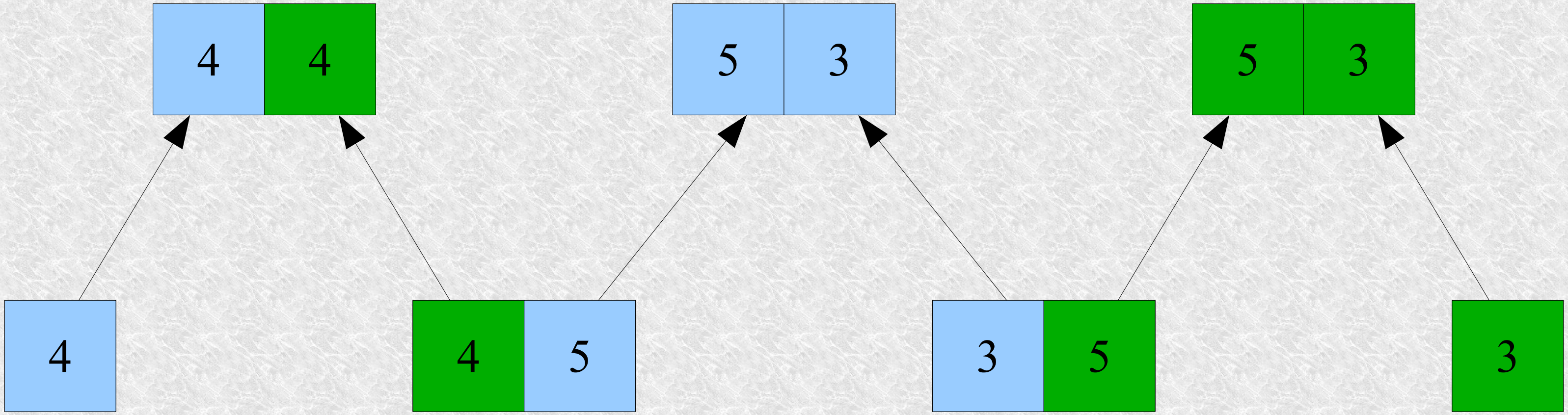
The outermost elements do nothing but return a number

After **2n** cycles we have the result ready

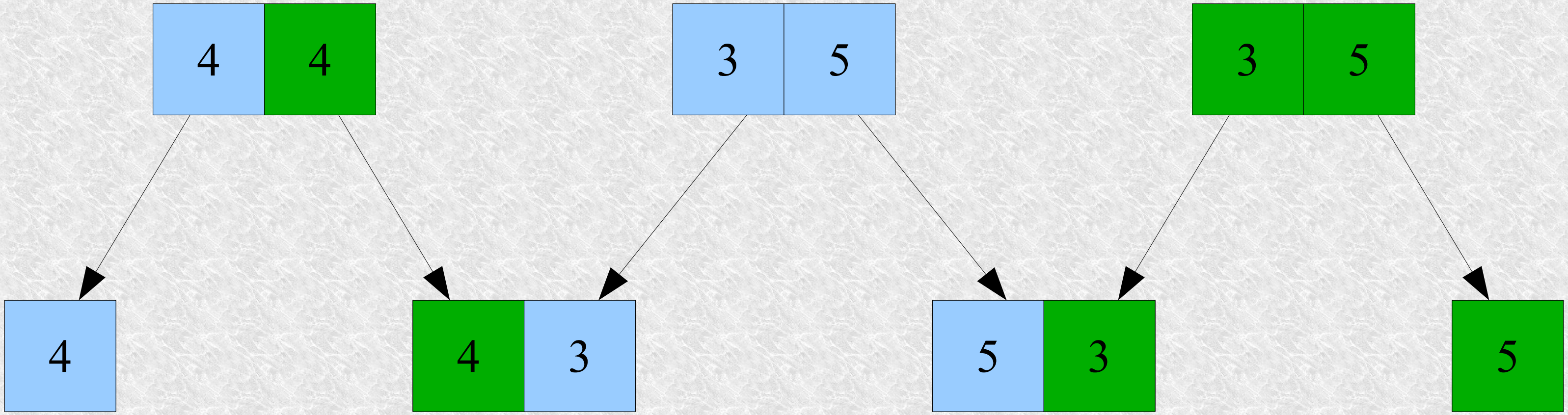
Oscillatory Sorting



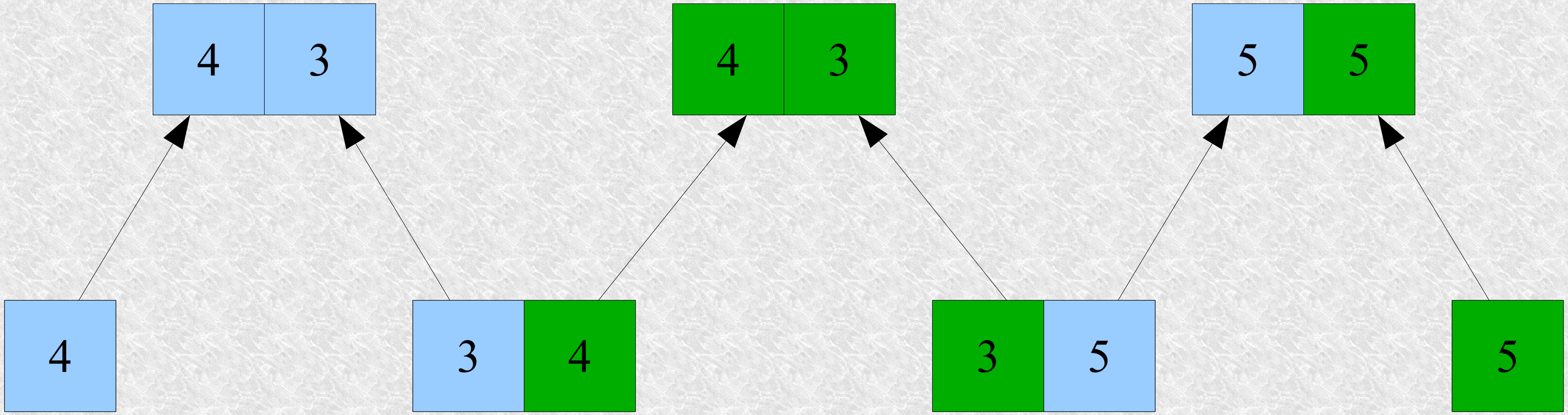
Oscillatory Sorting



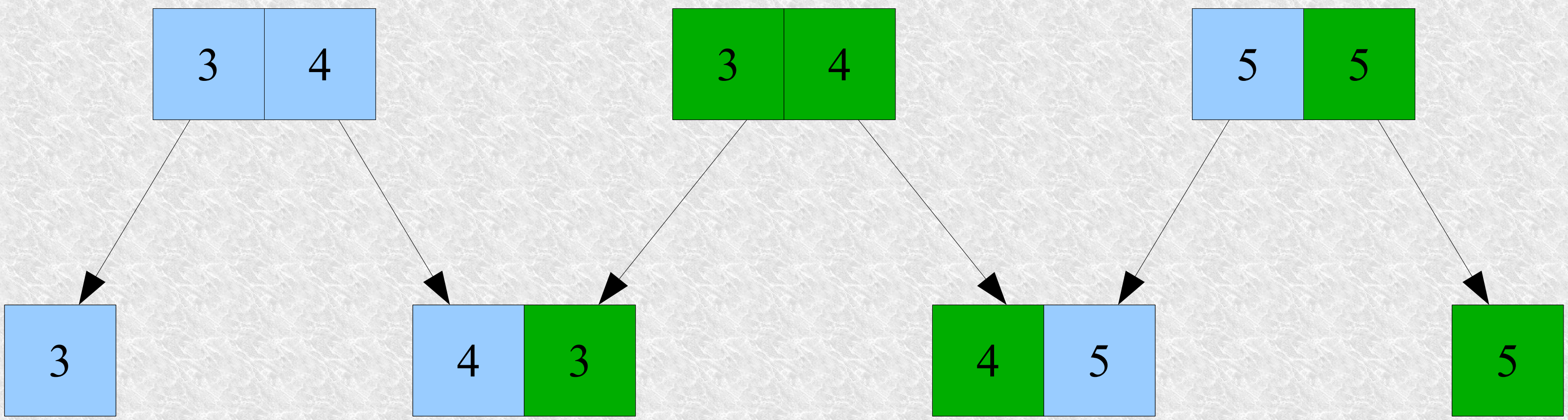
Oscillatory Sorting



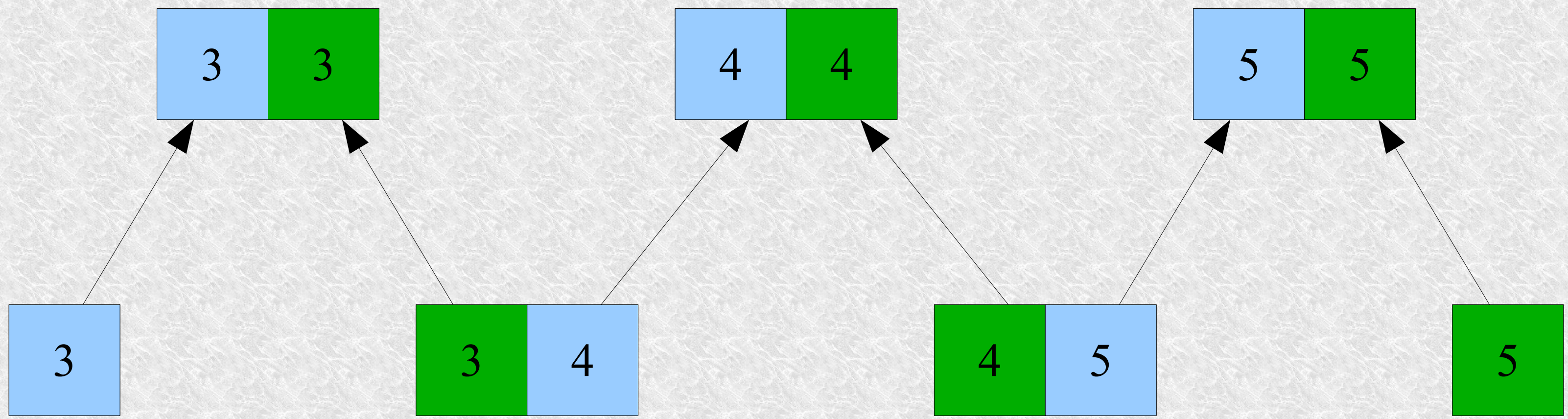
Oscillatory Sorting



Oscillatory Sorting



Oscillatory Sorting



Matrix multiplication

Application:

- in mathematics, solving systems of linear equations using Gauss's method
- Recording geometric objects in linear space in physics, so-called tensors
- Three-dimensional graphics, transformations

Matrix multiplication

Definition

The product of the matrix **A**=$[a_{ij}]_{nxp}$ by the matrix **B**=$[b_{ij}]_{pxm}$ is called such a matrix **C**=$[c_{ij}]_{nxm}$ we write **C=A•B**, that

$$c_{ij} = \sum_{k=1}^{p} a_{ik} \bullet b_{kj}$$ dla i=1,2,...,n;j=1,2,...,m

Matrix multiplication

$A \in M_{m \times k}$ $\qquad$ $B \in M_{k \times n}$ $\qquad$ $C \in M_{m \times n}$

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ & (\dots) & & \\ a_{m1} & a_{m2} & \dots & a_{mk} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ & (\dots) & & \\ b_{k1} & b_{k2} & \dots & b_{kn} \end{pmatrix} = \begin{pmatrix} \boxed{c_{11}} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ & (\dots) & & \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1k} \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{21} \\ (\dots) \\ b_{k1} \end{bmatrix} = \boxed{c_{11}}$$

Matrix multiplication

Some useful properties:

If A, B, and C are matrices of appropriate dimensions, then:

1. $A(BC)=(AB)C$
2. $(AB)=(A)B$
3. $(A+B)C=AC+BC$
4. $C(A+B)=CA+CB$
5. $IA=A$, gdy A_{nxn} i I_{nxn}

Matrix multiplication

„Divide and Conquer" Algorithm

Getting matrix **C** is the result of independent arithmetic operations on the rows of matrix **A** and the columns of matrix **B**. This makes it intuitive to divide the task into multiple threads, each independently computing an element of matrix **C**. In this case, there must be m*n such partitions (the number of threads). The cost of the operation is $\mathbf{O(n^3)}$.

Matrix multiplication

Every element A_{ij} B_{jk} C_{ik} represents a small submatrix, and the same computational operations are applied to it as to scalar elements.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Matrix multiplication

Example:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 4 & 1 & 2 \\ 0 & 2 & 2 & 1 \\ 1 & 2 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 2 & 0 \\ 2 & 3 & 1 & 2 \\ 1 & 1 & 2 & 3 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 8 & 5 & 11 \\ 9 & 16 & 8 & 13 \\ 6 & 9 & 6 & 11 \\ 5 & 10 & 6 & 9 \end{bmatrix}$$

Such multiplication can be divide into four operations analogous to the one below.

$$\begin{bmatrix} 0 & 1 \\ 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 8 & 13 \end{bmatrix} + \begin{bmatrix} 2 & 5 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 9 & 16 \end{bmatrix}$$

Matrix multiplication

Strassen's method

- From the partitioned matrix, compute seven auxiliary matrices **m_i** of size **n/2**

$m_1=(A_{12}-A_{22})*(B_{21}+B_{22})$
$m_2=(A_{11}+A_{22})*(B_{11}+B_{22})$
$m_3=(A_{11}-A_{21})*(B_{11}+B_{12})$
$m_4=(A_{11}+A_{12})*B_{22}$
$m_5=A_{11}*(B_{12}-B_{22})$
$m_6=A_{22}*(B_{21}-B_{11})$
$m_7=(A_{21}+A_{22})*B_{11}$

Matrix multiplication

Compute the components Cij of the resulting matrix C

$C_{11}=m_1+m_2-m_4+m_6$
$C_{12}=m_4+m_5$
$C_{21}=m_6+m_7$
$C_{22}=m_2-m_3+m_5-m_7$

The computational cost of the above algorithm is estimated as $\boldsymbol{O(n^{\log_2 7})}$

Matrix multiplication

Canon's Algorithm

- We assume a task grid of size m x m.
- Each process (t_{ij} where $0<i,j<m$) contains blocks C_{ij}, A_{ij} and B_{ij}.
- At the beginning of the algorithm, each process on the diagonal (i.e., t_{ij} where $i=j$) sends its block A_{ij} to all other processes in row i.
- After the transmission of A_{ii}, all tasks compute $A_{ii}xB_{ij}$ and add the result to C_{ij}.
- In the next step, the column of matrix blocks B is rotated. It means, each process t_{ij} sends its block to $t_{(i-1)j}$. Process t_{0j} sends its block B to $t_{(m-1)j}$.

Matrix multiplication

- Now the processes return to the first step.
- The block $A_{i(i+1)}$ becomes the fundamental piece of information for all other processes in row i.

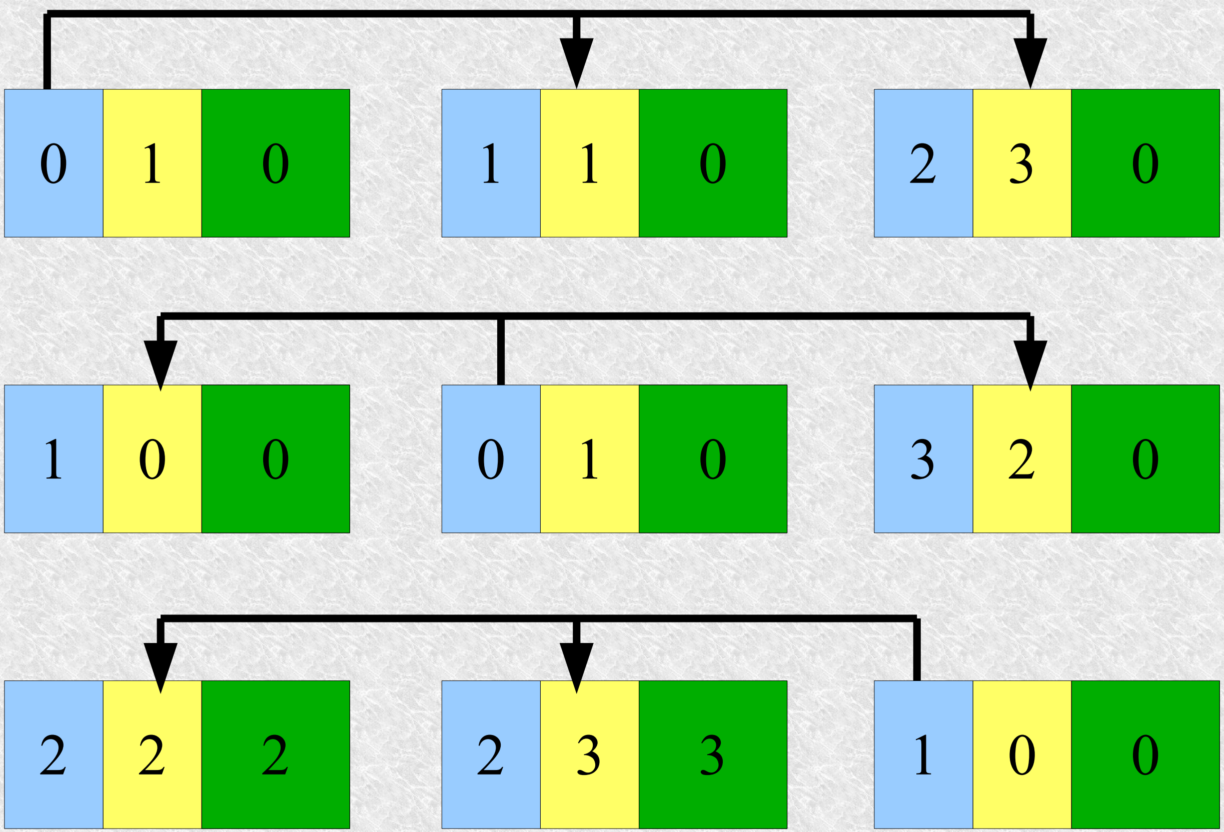
- The algorithm continues.
- After m iterations, the matrix C contains the result of the multiplication AxB, and the rotated matrix B returns to its original configuration.

Matrix multiplication

$$\underset{A}{\begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 2 & 1 \end{bmatrix}} \cdot \underset{B}{\begin{bmatrix} 1 & 1 & 3 \\ 0 & 1 & 2 \\ 2 & 3 & 0 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 4 & 7 & 2 \\ 7 & 10 & 3 \\ 4 & 7 & 10 \end{bmatrix}}$$

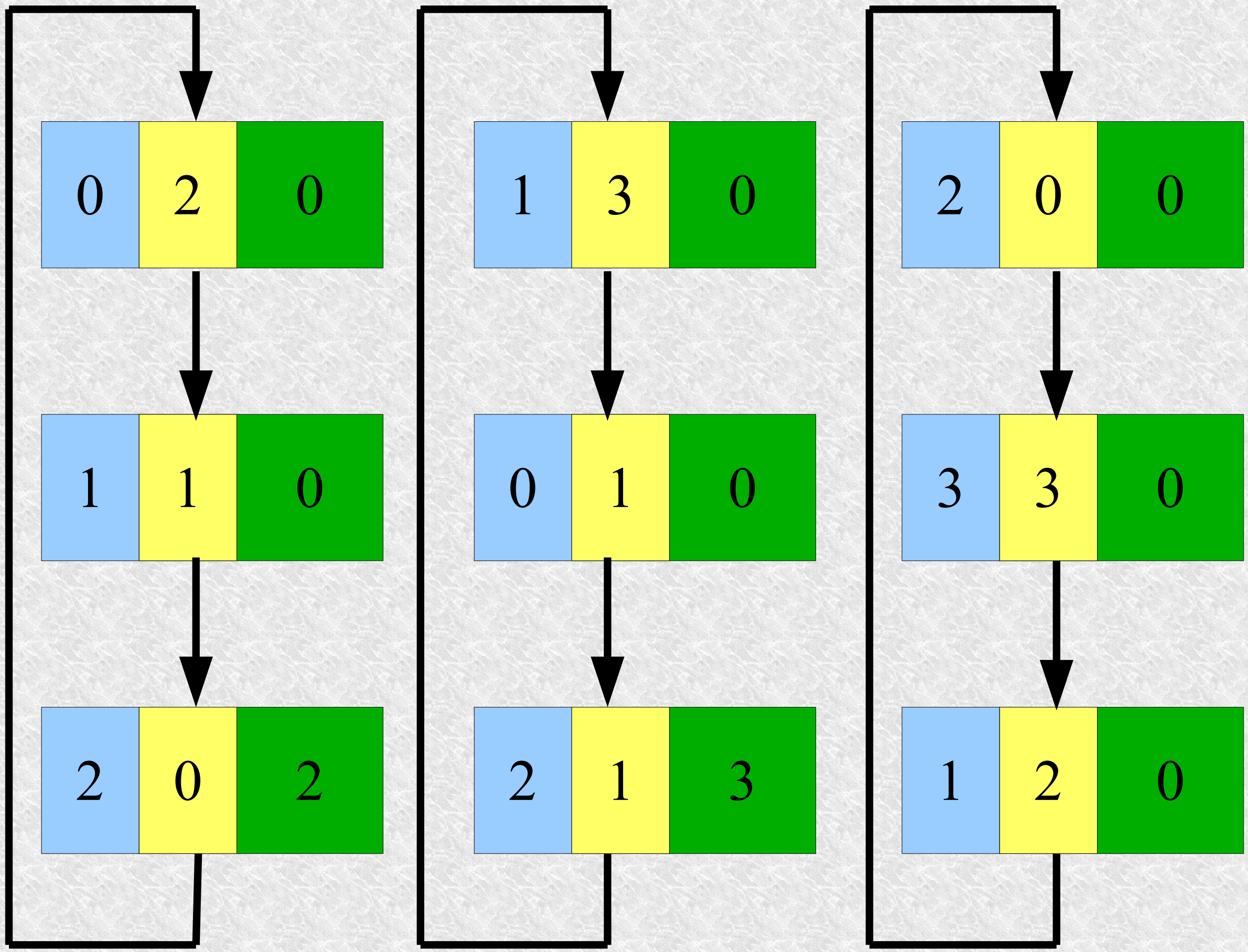
0	1	0	1	1	0	2	3	0
1	0	0	0	1	0	3	2	0
2	2	0	2	3	0	1	0	0

Matrix multiplication



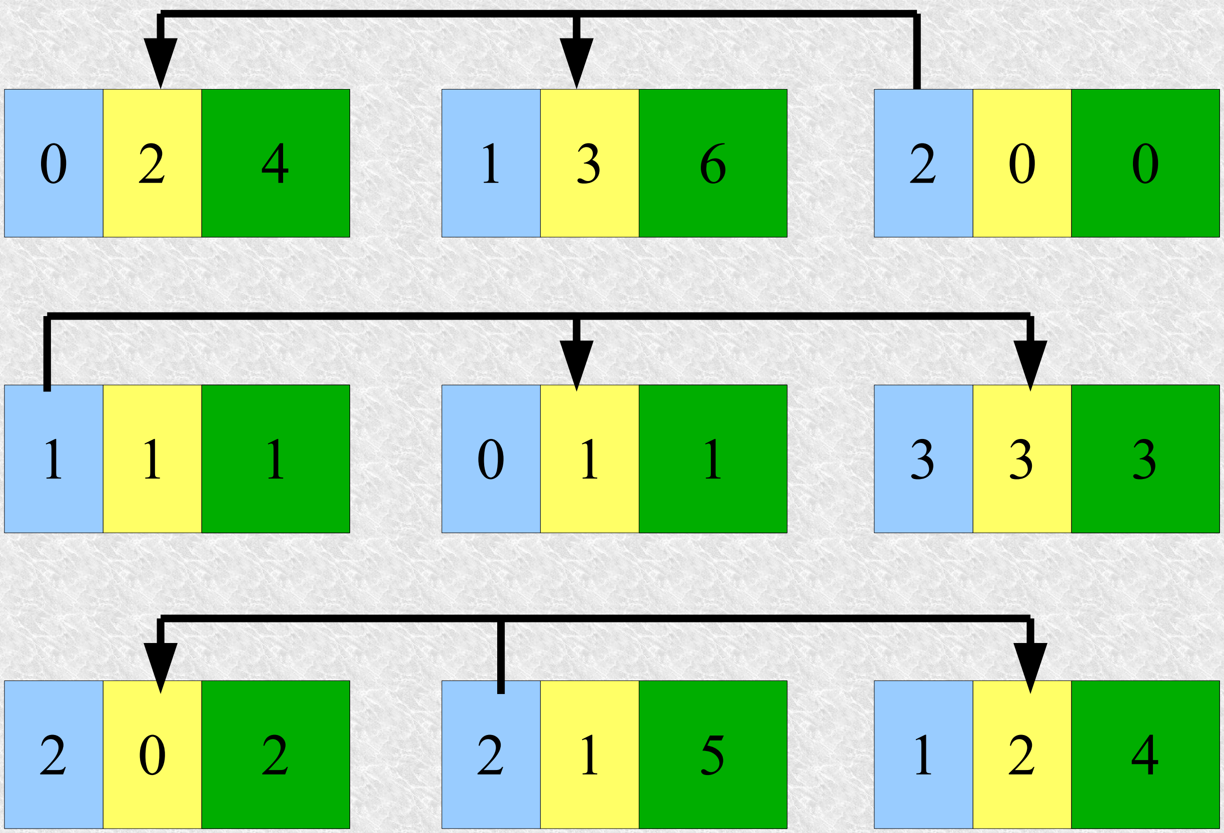
From matrix A, we take the values located on the diagonal. These values are sent to neighboring processes in the same row. The received values are multiplied by the corresponding values from matrix B, and the result is added to the partial result in matrix C.

Matrix multiplication



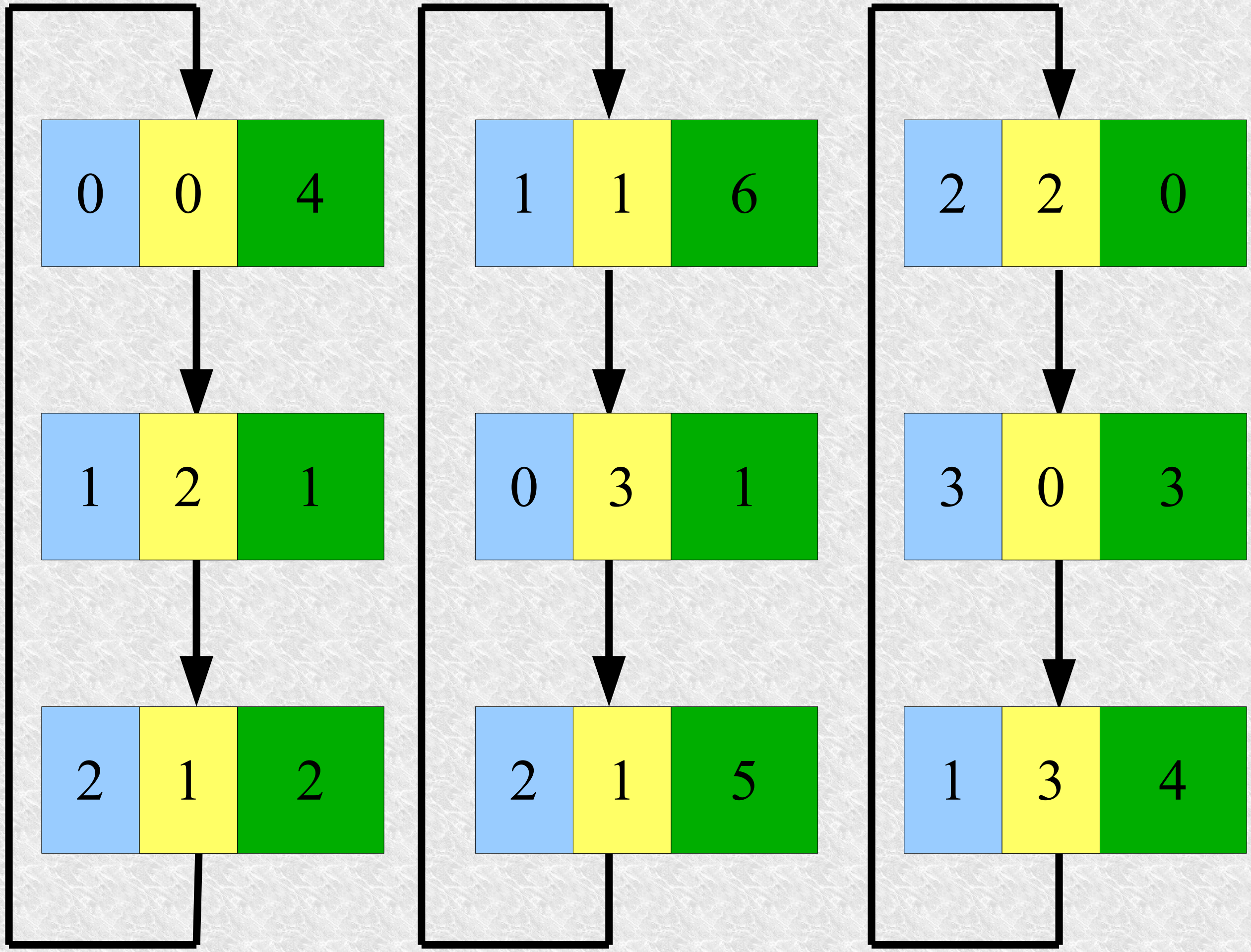
The columns of matrix *B* are cyclically shifted downward.

Matrix multiplication



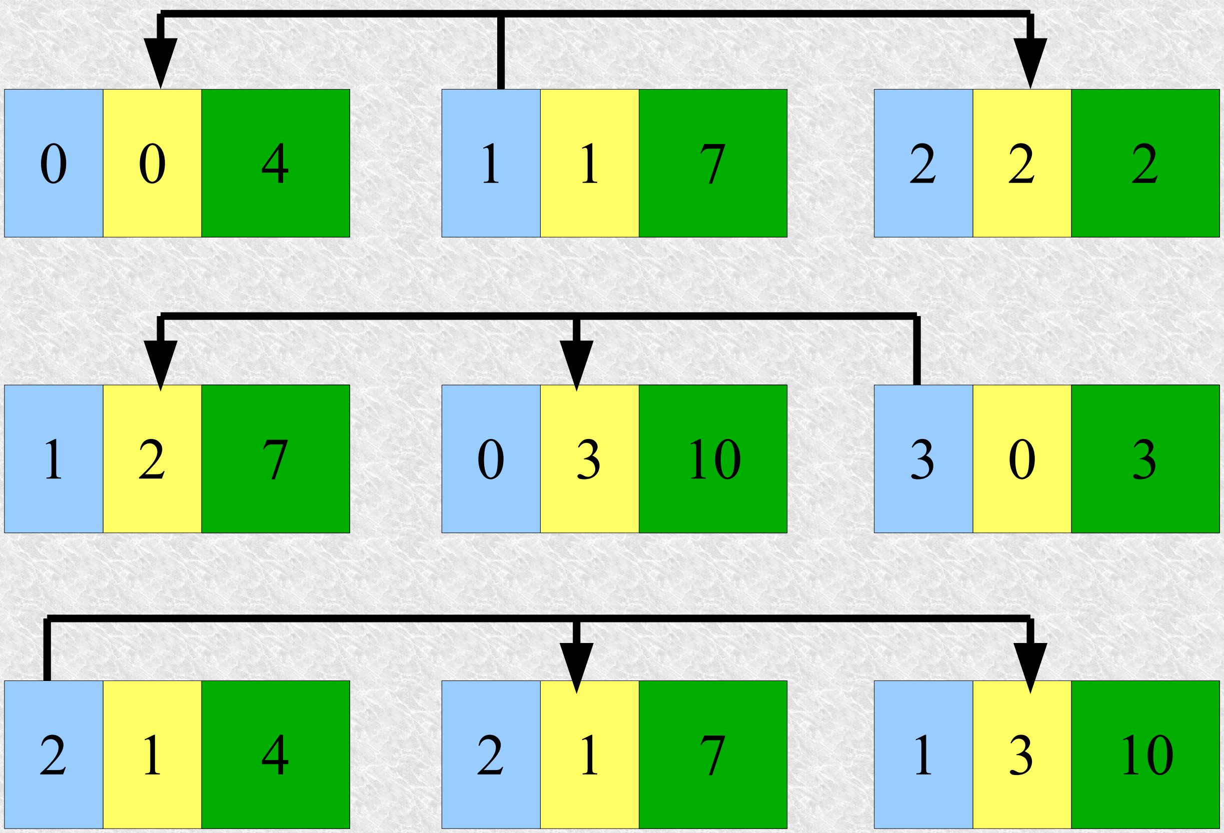
We shift the diagonal one row downward. The new diagonal values are sent to neighboring processes in the same row. These transmitted values are multiplied by the corresponding blocks of matrix B, and the result is added to the partial result in matrix C.

Matrix multiplication



The columns of matrix *B* are cyclically shifted downward.

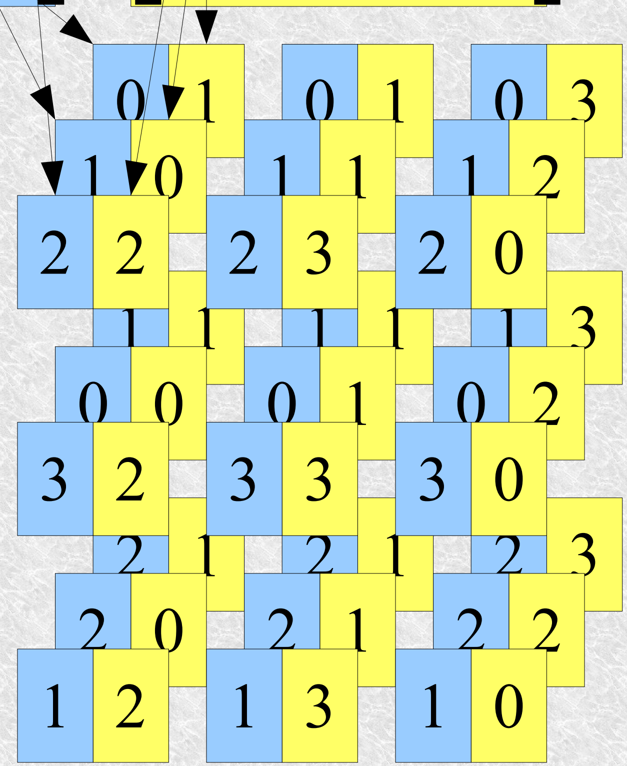
Matrix multiplication



We shift the diagonal one row downward. The new diagonal values are sent to neighboring processes in the same row. These transmitted values are multiplied by the corresponding blocks of matrix B, and the result is added to the partial result in matrix C.

3D matrix multiplication

$$\overset{A}{\begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 2 & 1 \end{bmatrix}} \cdot \overset{B}{\begin{bmatrix} 1 & 1 & 3 \\ 0 & 1 & 2 \\ 2 & 3 & 0 \end{bmatrix}} = \overset{C}{\begin{bmatrix} 4 & 7 & 2 \\ 7 & 10 & 3 \\ 4 & 7 & 10 \end{bmatrix}}$$



Genetic Algorithm

- A genetic algorithm is a type of algorithm that searches the space of alternative solutions to a problem in order to find the best ones.
- It belongs to the class of evolutionary algorithms.
- The concept was introduced by John Henry Holland.

Source: wikipedia

Genetic Algorithm

- Environment defines the problem
- Population – a set of individuals
- Genotype – information assigned to an individual
- Phenotype – traits evaluated by the fitness function
- Fitness function – models the environment
- Chromosomes – components of the genotype
- Genes – elements of chromosomes

Source: wikipedia

Genetic Algorithm

Common Key Features

- Use of genetic operators like crossover and mutation
- Parallel search from multiple starting points
- Search direction guided by solution quality
- Intentional randomness introduced to avoid local optima

Source: wikipedia

Genetic Algorithm

Basic Algorithm Steps

- Randomly generate initial population
- Develop individuals and evaluate fitness
- Check exit condition
- Selection (better individuals have higher chance)
- Apply genetic operators:
  - Crossover
  - Mutation
- Create new generation (there algorithm repeats from step 2)

Source: wikipedia

Genetic Algorithm

Encoding

- Chromosome as vector of genes
- Genes can be binary, integer, or real numbers
- Logarithmic encoding
- Tree structures also possible

Source: wikipedia

Genetic Algorithm

Evaluation

- Chromosomes are input to a function f(x)
- The value of f(x) indicates how well-adapted the individual is x
- Optimization seeks the maximum or minimum of f(x)

Source: wikipedia

Genetic Algorithm

Selection Methods

- Roulette wheel
  - The entire roulette wheel corresponds to the sum of the fitness function values for all chromosomes in the population.
  - For each chromosome ch_i, for $i=1,2,...,N$, where N is the population size, a segment of the wheel is assigned according to the formula $$Ch_i = p_s * 100\%$$

    The probability is expressed as:

$$p_s(ch_i) = \frac{F(ch_i)}{\sum_{i=1}^{N} F(ch_i)} \; przy \, (i = 1, 2, ..., N)$$

  - $F(ch_i)$ is the fitness function value of chromosome ch_i
  - The selection (spinning) is performed N times

Genetic Algorithm

Natural Selection Methods

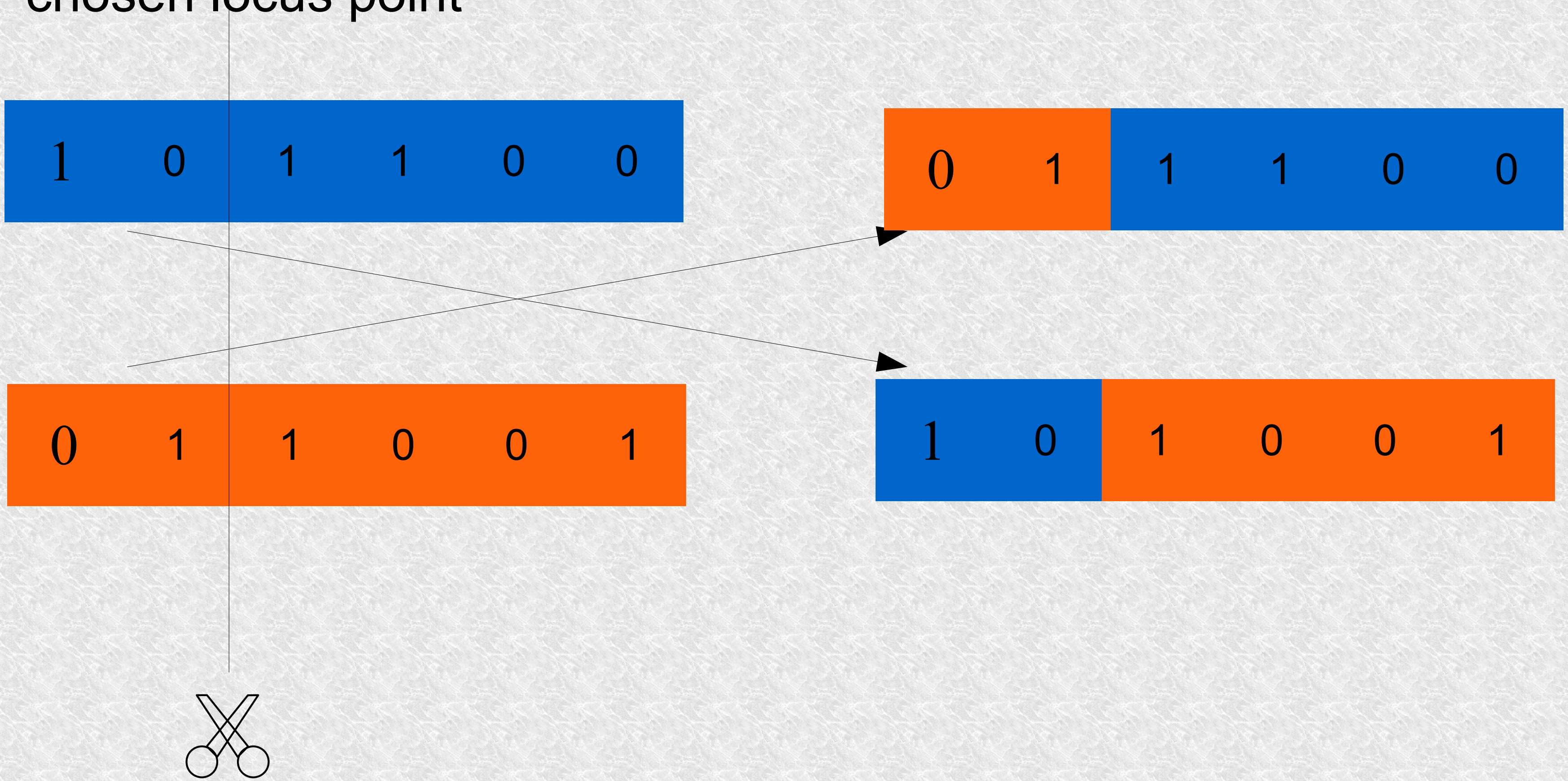
- **Elitist Method**
  - The best individual is passed on unchanged to the next generation.
- **Tournament Method**
  - Individuals are divided into groups, and the best solution from each group is selected.
- **Ranking Method**
  - Solutions are sorted based on their quality and assigned ranks. Then, a linear function is typically used to select an appropriate number of individuals, starting from the best-ranked.

Source: wikipedia

Genetic Algorithm

Crossover

- The basic crossover method involves splitting the chromosome at a chosen locus point



Źródło: wikipedia

Genetic Algorithm

Other Crossover Methods

- Partially Mapped Crossover (PMX)
  - Preserves relative ordering and position of genes by partially mapping segments between parents. (TSP)
- Order Crossover (OX)
  - Maintains the relative order of genes from one parent while filling in the remaining genes from the other parent. (queue optimization, production planing)

Genetic Algorithm

Example

- Finding the maximum of the function $f(x)=x^2$ in the range 0..31
- We choose a 5-bit encoding system for the decision variable x.
- The population consists of 4 binary strings.

01101
11000
01000
10011

Genetic Algorithm

Example

#	Initial Population	Value x	f(x)	Probability	Expected Copies	Randomly Selected Copies	Parent Pool	Random choosen Partner	Crossover Point	New Generation	x	f(x)
1	01101	13	169	0,14	0,58	1	011|01	2	4	01100	12	144
2	11000	24	576	0.49	1,92	2	110|00	1	4	11001	25	625
3	01000	8	64	0,06	0,22	0	11|000	4	2	11011	27	729
4	10011	19	361	0,31	1,23	1	10|011	3	2	10000	16	256
Sum			1170	1,00	4	4						1754
Avg			293	0,25	1	1						439
Max			576	0,49	1,97	2						729

Genetic Algorithm

Application in Concurrent Programming

- **Master-slave architecture**
  - The master process delegates the development of the population to worker processes. Each worker (slave) may handle an entire group or a single individual. Slaves return the fitness function values to the master. The master is responsible for performing natural selection.
- **Peer-to-peer (P2P) architecture**
  - Natural selection is performed without a central coordinating process. Chromosomes are exchanged directly between processes.

Ant Colony Optimization

A Few Words About Ants

- Ants are practically blind.
- They possess an excellent sense of smell.
- Their brains are extremely small.
- When acting collectively, they can find the shortest path between food and the nest.
- They rely on pheromones and the phenomenon of stigmergy — a form of indirect communication through environmental changes that others can detect and respond to.
- As ants march, they leave pheromone trails behind.
- Pheromones continuously evaporate over time.
- When deciding which path to follow, ants are guided by the intensity of the pheromone scent.

Ant Colony Optimization

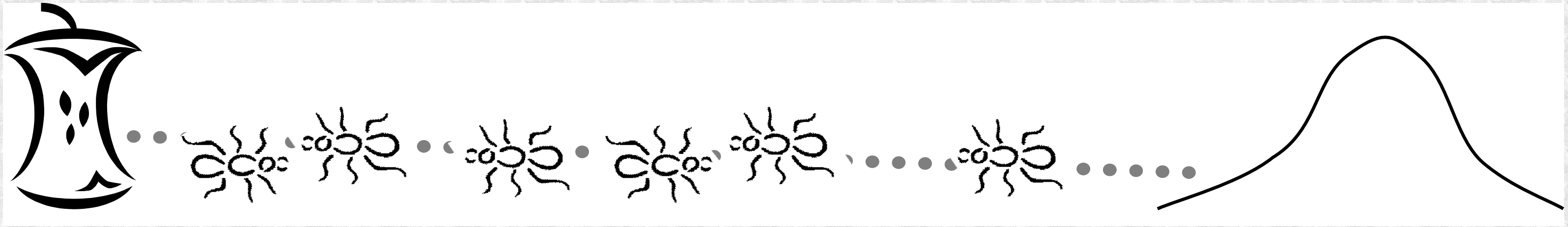
Algorithm Behavior

- Ants randomly disperse from the nest in search of food.
- Those that find shorter paths return more quickly, retracing their own pheromone trails.
- On shorter paths, the number of ants per unit of time is higher, which leads to more frequent pheromone reinforcement.
- Longer paths, being visited less often, experience pheromone evaporation and gradually disappear.

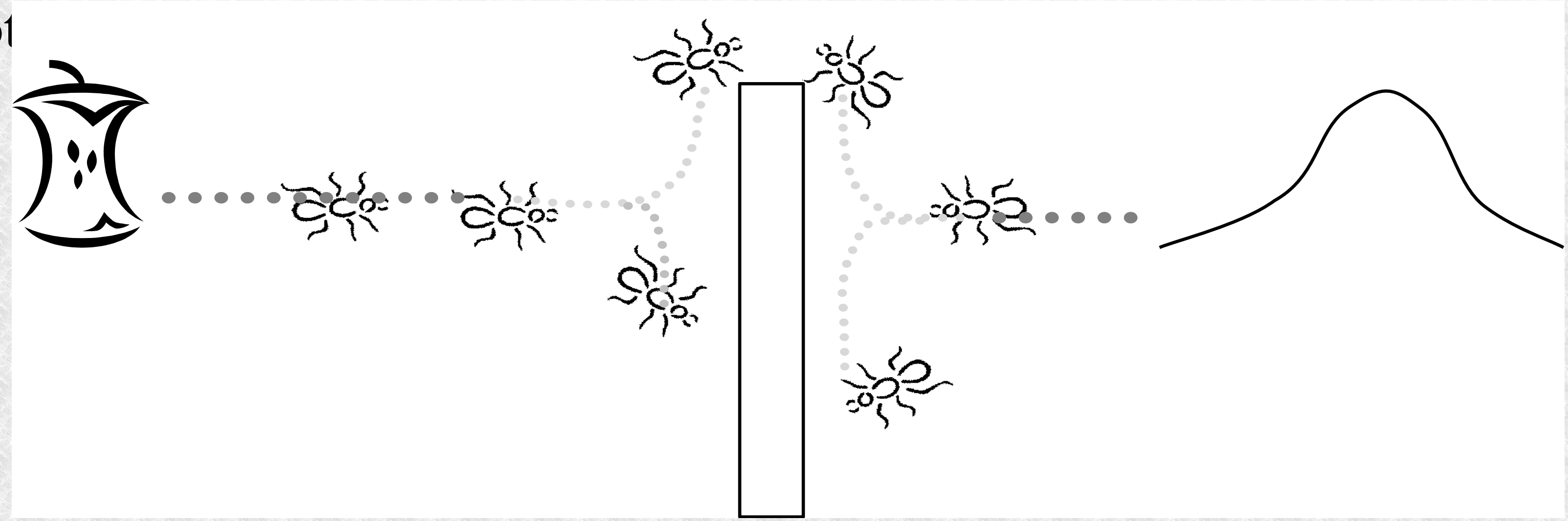
Ant Colony Optimization

Decision-Making in the Algorithm

- Ants follow a predefined path.



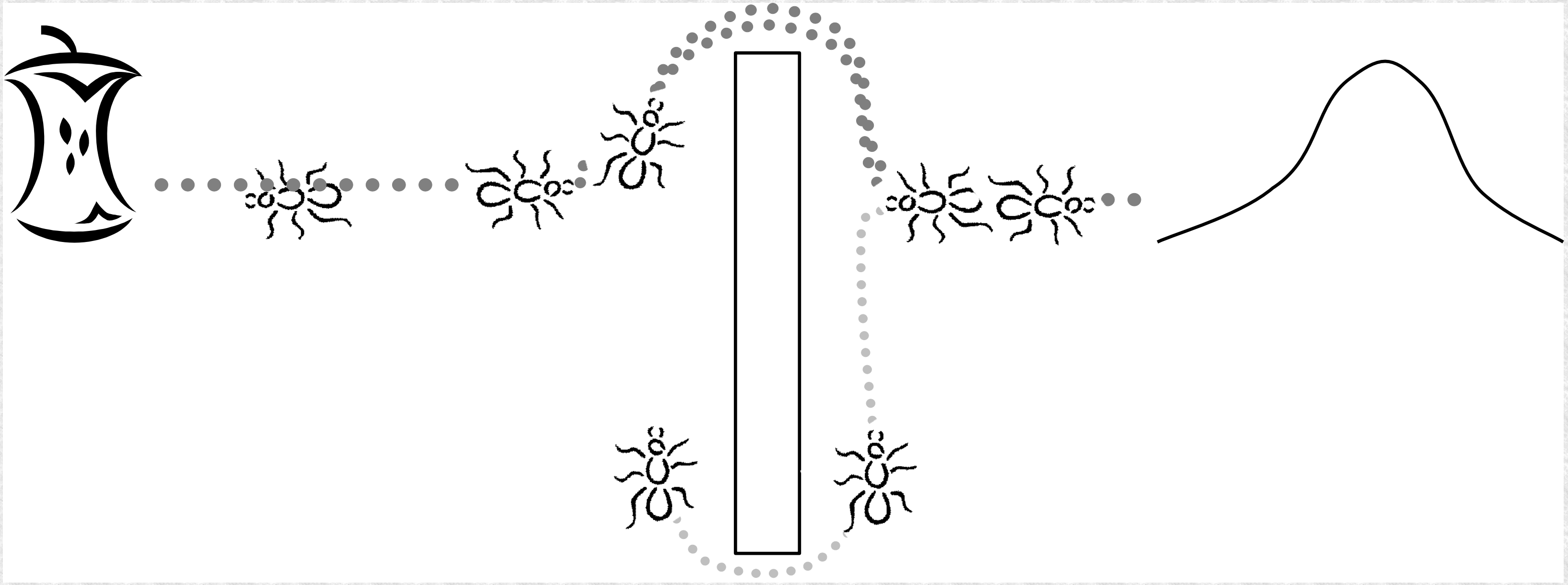
- When ants encounter an obstacle, some go around it from one side, while ot



Ant Colony Optimization

Decision-Making in the Algorithm

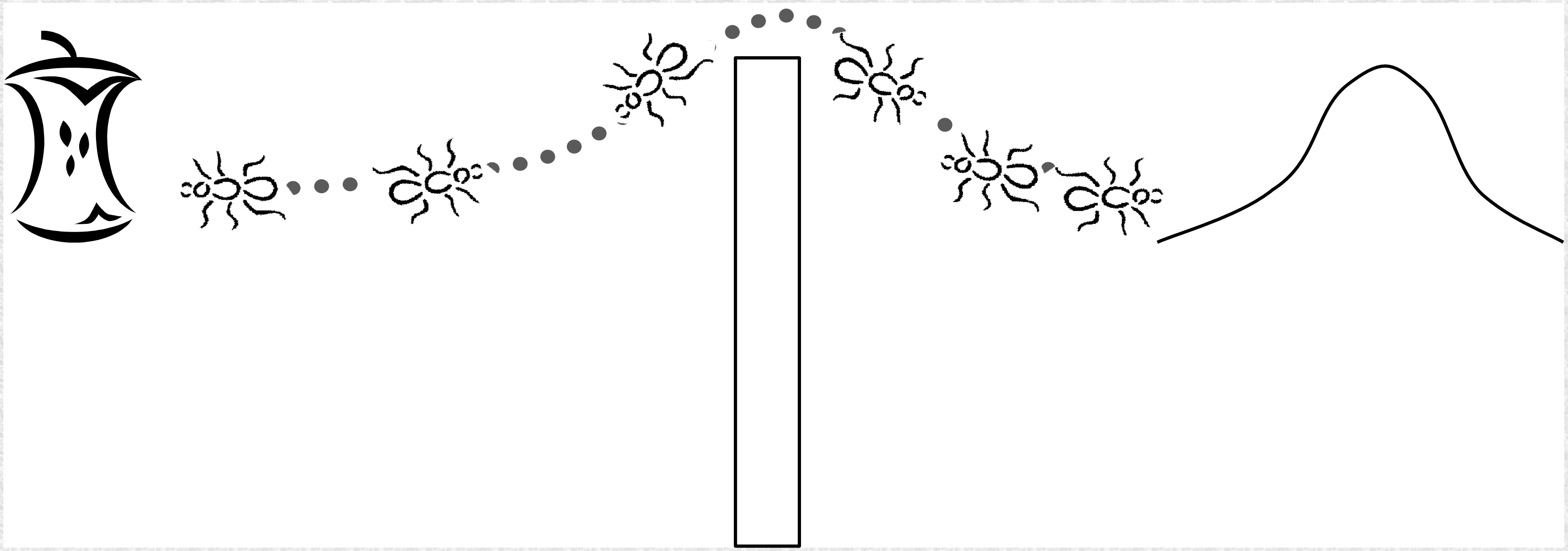
- Ants that follow the shorter path return faster and reinforce the pheromone trail sooner. This creates a stronger signal for other ants, indicating that this route is more optimal.



Ant Colony Optimization

- Decision-Making in the Algorithm

- As the algorithm progresses, ants gradually converge toward the shortest and most efficient path.



Ant Colony Optimization

Finding the Optimal Route in the Traveling Salesman Problem (TSP)

- Starting point – arbitrary
- Goal of a single ant – visit all cities exactly once
- Movement – ants travel along the edges of a graph
- Iterative behavior – ants construct solutions in multiple iterations
- Pheromone update – applied only after all ants have completed their tours in a given iteration (Cycle Ant System – CAS)
- Pheromone quantity – either:
    - Inversely proportional to the length of the path found, or
    - Deposited only by the best-performing ant

Ant Colony Optimization

Finding the Optimal Route in the Traveling Salesman Problem (TSP)

- Initially, each edge has an equal amount of pheromone.
- Each ant selects a random starting point.
- When choosing the next i-th node in its tour, the ant is guided by a decision table that combines pheromone intensity and heuristic information.

$$A_i = [a_{ij}(t)]_{[N_i]}$$

Ant Colony Optimization

Elements of the Decision Table Are Defined by the Formula:

$$a_{ij} = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} \forall j \in N_i$$

- τ_{ij} - pheromone intensity on edge $i \rightarrow j$
- $\eta_{ij} = \frac{c}{d_{ij}}$ The heuristic value indicating the attractiveness of the connection between point i and point j. c tis a constant d is the distance between nodes i and j
- N_i- – the set of all unvisited nodes directly reachable from node i
- The parameters α and β are used to adjust the relative importance of pheromone intensity and heuristic information when ants choose their next move. A higher α emphasizes pheromone trails, while a higher β gives more weight to heuristic desirability (e.g., shorter distances).

Ant Colony Optimization

The probability that ant k, currently at node i, will choose node $j \in N_i^k$ during iteration t is given by:

$$p_{ij}^k(t) = \frac{a_{ij}(t)}{\sum_{l \in N_i^k} a_{il}(t)}$$

Where $N_i^k \subseteq N_i$ set of feasible (unvisited) nodes for ant k from node i.

Ant Colony Optimization

Once all ants have found a path, we select the best one or proportionally update the pheromone amount on each edge of the route by adding m/L of pheromone, where m is a constant and L is the length of the found path. (CAS Strategy).

Evaporation is also involved. We evaporate a constant portion of the pheromone from each edge of the graph:

$$\tau_{ij} = \tau_{ij} * \rho$$

Ant Colony Optimization

In the context of concurrent processing, each ant can be a separate thread. Synchronization should guarantee synchronization between subsequent stages when updating the pheromone map